TR 03-029

PAFI: A Pattern Finding Toolkit

Masakazu Seno, Michihiro Kuramochi, and George Karypis

July 07, 2003

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **07 JUL 2003**	2. REPORT TYPE	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **PAFI: A Pattern Finding Toolkit**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Army High Performance Computing Research Center,Department of Computer Science and Engineering,University of Minnesota,Minneapolis,MN,55455**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT **see report**		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**21**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

PAFI [*]
A Pattern Finding Toolkit

Release 1.0.1

Masakazu Seno, Michihiro Kuramochi, and George Karypis

Department of Computer Science, University of Minnesota
Minneapolis, MN 55455

{seno, kuram, karypis}@cs.umn.edu

Last updated on July 7, 2003 at 12:52am

[*]PAFI is copyrighted by the regents of the University of Minnesota. This work was supported by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Related papers are available via WWW at URL: **http://www.cs.umn.edu/˜karypis**. The name PAFI is derived from PAttern FInding toolkit.

Contents

1 Introduction

PAFI is a set of programs that can be used to find frequent patterns in large and diverse databases. The current release of PAFI includes three different pattern discovery programs called LPMiner, SLPMiner, and FSG. LPMiner finds patterns corresponding to itemsets in a transaction database and is based on the algorithm described in [5]. SLPMiner finds patterns corresponding to sub-sequences in a sequential database and is based on the algorithm described in [6]. Finally, FSG finds patterns corresponding to connected undirected subgraphs in an undirected graph database and is based on the algorithms described in [3, 4]. These programs can be used to mine a wide-range of datasets arising in commercial, information retrieval, and scientific applications [1].

All three programs can be used to find patterns that satisfy a constant minimum support. Moreover, a key feature of LPMiner and SLPMiner is that they can find long frequent patterns without finding a large number of short patterns that are often useless. This is achieved by using *length-decreasing support constraints*, where the minimum occurrence frequency of a pattern is given as a non-increasing function of pattern length.

PAFI's pattern discovery programs usually provide three additional functionalities. First, all three programs can generate maximal frequent patterns. A maximal frequent pattern is a frequent pattern that is not contained by any other frequent patterns. Generally, the number of maximal frequent patterns is much smaller than the number of all the frequent patterns, leading to higher readability of frequent pattern files. Second, SLPMiner and FSG can generate *transaction-ID lists* (TID-lists) indicating which sequences or graph transactions support a particular frequent pattern. Third, all three programs can generate *parent-children-lists* (PC-lists) that can be used to construct the frequent pattern lattice.

Outline of PAFI's Manual

PAFI's manual is organized as follows. Sections 2, 3, and 4 describe the command-line options of the different frequent pattern discovery programs and the format of the input and output files that they require and generate. Section 5 describe the general filename rules associated with the various files generated by PAFI's programs. Finally, Section 6 describes the system requirements for the PAFI package and provides contact information.

2 The LPMiner Program

Usage

LPMiner [optional parameters] *TranFile*

Description

Used to find all frequent itemsets satisfying a constant or length-decreasing support constraint from a transaction file.

Required Parameters

TranFile The name of the file that stores the input transactions from which frequent itemset patterns are to be found. The format of the transaction file is described in Section 2.1.1.

Optional Parameters

`-s FLOAT, --support=FLOAT`

This parameter sets the constant minimum support in percentage. This value must be in the range of [0.0, 100.0]. The default value is 5.0%. Note that if the minimum support is set to 0.0, all the frequent patterns with at least one supporting transaction are output. If *-S string* option is specified, -s float option is ignored.

`-S FILE, --supptable=FILE`

This parameter specifies the file that stores the length-decreasing support constraint. The format of this file is described in Section 2.1.2. This parameter hides *-s float* option.

`-m INT, --minsize=INT`

This parameter sets the minimum length of frequent patterns to be output. The default value is one. If this parameter is set to a value greater than one, the amount of time required to find the frequent patterns will be decreased.

`-M INT, --maxsize=INT`

This parameter sets the maximum length of frequent patterns to be output. The default value is 4,294,967,295 (0xffffffff). If this parameter is set to a smaller value, then the amount of time taken for finding frequent patterns may be decreased.

`-x, --maximal`

Generates maximal frequent patterns only. The current version of LPMiner does not contain any optimizations to actually reduce the amount of time required to find maximal patterns, and this parameter is used solely to limit the amount of information that is being output.

`-p, --parent-children-list`

Generates a file that shows the parent-children relationships among frequent patterns. These relationships are referred to as *PC-list*. For each frequent pattern, p, its PC-list contains all frequent patterns c, such that c is a maximal frequent sub-pattern of p. Note that when LPMiner is used with a constant support constraint, the PC-lists correspond to the frequent pattern lattice.

The PC-lists are stored in a file, which has the same name as the input file with file extension ".pc" added. The format of the PC-list file is described in Section 2.2.2.

`-h, --help`

Displays a short summary of the various options to the standard output.

Output

The discovered frequent patterns are stored in a file that has the same name as the input file with file extension ".fp" added. The format of this frequent pattern file is described in Section 2.2.1.

The LPMiner program also generates and prints various statistics regarding the input file, the discovered frequent patterns, and the amount of time taken during the various stages of the computation. The same information is also appended to a file called `lpminer.log`.

NOTE

The LPMiner program uses an in-memory implementation. For this reason, the size of the datasets that can be processed are limited by the amount of physical memory in your system. If that is not enough, you can use the SLPMiner program to find frequent itemset patterns, as it is disk-based. However, the runtime of SLPMiner is an order of magnitude higher than LPMiner.

2.1 Input File Formats

LPMiner takes as input two different files. The first is the *TranFile* that contains the transactions from which the frequent patterns will be discovered, and the second file is the length-decreasing support constraint file (*SuppFile*) that indicates the different minimum support values associated with each pattern length. Note that the *SuppFile* is optional and is used only when the `-S` is specified. The format of these files is described in the rest of this section.

2.1.1 Transaction File

A transaction file is an ASCII file in which each line represents a transaction. A transaction is a set of items separated by one or more spaces or tabs. Each item can be any string except -1. There are no restrictions on how the items are ordered within each transaction. Figure 1 shows some examples of different transaction files.

```
1 2 4 6 7 9 11
1 2 3 4 6 7 8 9 12 13 15 17
5 7 9 10 11 13 15 18
1 2 4 5 16 19 20
0 1 2 4 8 9 11 13 15
0 1 2 6 12 14 19
2 3 6 8 9 10 12 14 15 16 19
2 4 6 7 8 11 12 14 16 17
1 2 3 4 5 8 10 14 18
0 1 2 4 6 9 11 13
```

(a)

```
apple orange pear melon
orange pear
orange melon peach
apple orange grape melon
```

(b)

```
item1 item3 item6 item4
item3 item6 item4 item1
item3 item2 item7
item1 item4 item7
```

(c)

Figure 1: Examples of transaction files accepted by LPMiner. Note that the items can be arbitrary strings.

If a line contains nothing except spaces or tabs, the line is ignored and it is not counted when determining the total number of transactions. Also, if a line contains duplicate items, then the duplicate occurrences of an item are ignored. Ordering of the items within each transaction does not affect the arrangement of items in frequent pattern files generated by LPMiner.

2.1.2 Length-Decreasing Support Constraint File

A length decreasing support constraint is a non-increasing function of pattern length, $f(m)$ $(m = 1, 2, \ldots)$. The *length* of a pattern is the number of items in the pattern. A pattern with length m is frequent if and only if its support is at least $f(m)$%. The *support* of a pattern (in percentage) is defined as $100m/n$, where m is the number of transactions supporting the pattern and n is the number of all the transaction in the input transaction file.

A length-decreasing support constraint is specified by an ASCII file. Each line of the file contains a pair of numbers m and $f(m)$ separated by one space. The number m is an integer greater or equal to one and $f(m)$ is a floating point number $(0.0 \leq f(m) \leq 100.0)$ indicating the minimum support for patterns of length m. Note that it must always hold that $f(m_1) \geq f(m_2)$ for any two integers m_1 and m_2 such that $m_1 < m_2$. All lines must be sorted in the ascending order of m. The first line of the file must always specify the minimum support for $m = 1$. Also, for patterns whose length is greater than the largest m value specified in the file ($m_{\max}$), LPMiner finds these longer patterns using a minimum support corresponding to $f(m_{\max})$.

Figure 2(a) shows an example of length-decreasing support constraint file. Given this length-decreasing support

```
1 5.0
2 5.0
3 5.0
4 4.5
5 4.5
6 4.5
7 4.0
8 4.0
9 4.0
```

(a) An example of length−decreasing support constraint file.

```
1 5.0
4 4.5
7 4.0
```

(b) Shortened length−decreasing support constraint file.

Figure 2: Examples of length-decreasing support constraint files.

constraint file, LPMiner uses 4.0% as the minimum support for patterns of length greater than 9. If for a range of pattern-lengths, the associated minimum support values are the same, then all but the first occurrence of the identical supports can be omitted. For example, the length-decreasing support constraint file in Figure 2(a) can be shortened to that shown in Figure 2(b).

In addition, you can specify more than 100.0 as $f(m)$ value. In that case, no frequent patterns with length m will be generated. Another special case is when you specify 0.0 as $f(m)$ value. In that case, LPMiner outputs every pattern that is supported by at least one transaction.

2.2 Output File Formats

Upon successful execution, LPMiner outputs the discovered patterns into a file called the *frequent pattern file*. If the `-x` option has been specified, then this file contains only the maximal patterns, otherwise it contains all patterns that satisfy the minimum support constraint(s). In addition, if the user has specified the `-p` option, LPMiner will generate an additional file called the *PC-list file*, that contains the parent-children relationships among frequent patterns. The format of these files is described in the rest of this section.

2.2.1 Frequent Pattern File

The frequent-pattern file (*.fp) stores either the frequent or the maximal frequent patterns discovered by LPMiner. The patterns in this file are sorted in increasing order of their pattern length. Each line has the following format:

```
PATTERN_ID FREQ SUPP PATTERN
```

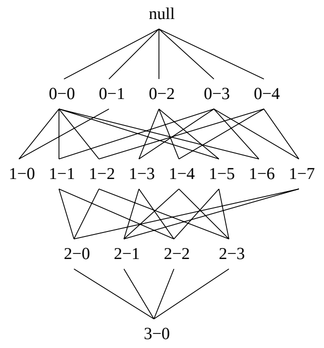
`PATTERN_ID` is a unique identifier of the pattern and is given as a *LEVEL-NODE* pair, where *LEVEL* is equal to the length of the pattern minus one, and *NODE* is a unique number associated with that pattern of that particular LEVEL. The NODE-numbers range from zero to the number of patterns of that LEVEL minus one. The pattern's `PATTERN_ID` is used to relate the frequent pattern with records in other output files produced by LPMiner. `FREQ` is the number of supporting sequences and `SUPP` is the same value in percentage. `PATTERN` is the frequent pattern in the same format as the input transaction file. All the items in each itemset of `PATTERN` is sorted so that integers come first arranged by the numerical ordering, and then non-integer strings follow arranged by the lexicographic ordering. Figures 5(b) and 5(d) shows two examples of frequent pattern files produced by LPMiner.

2.2.2 PC-List File

A *parent-children list* (PC-list) represents parent-children relationships among the patterns and can be used to construct the lattice of frequent patterns. The precise meaning of what constitutes the *children patterns* of a particular *parent pattern* depends on whether or not the patterns where discovered using a constant or a length-decreasing support constraint. If constant support was used, then for each parent pattern of size k, its children patterns correspond to all of its sub-patterns of size $k-1$. However, if a length decreasing support constraint was used, then for each parent

```
3−0 2−1 2−0 2−3 2−2
2−0 1−7 1−2 1−1
2−1 1−7 1−4 1−3
2−2 1−3 1−1 1−5
1−0 0−1 0−0
1−1 0−3 0−0
1−2 0−4 0−0
1−3 0−3 0−2
1−4 0−4 0−2
1−5 0−2 0−0
1−6 0−3 0−0
1−7 0−4 0−3
0−0
0−1
0−2
0−3
0−4
```

(a) An example PC−list



(b) It's corresponding pattern lattice

Figure 3: PC-lists and Pattern Lattices.

pattern of size k, its children patterns correspond to all of its maximal frequent sub-patterns of length less than k. Note that this distinction is due to the fact than not all patterns in the traditional itemset lattice will satisfy the given length-decreasing support constraint.

The format of the PC-list file (*.pc) is as follows. It contains as many lines as the number of frequent patterns, and for each parent pattern it lists its children patterns in the following format:

`<PARENT_PATTERN_ID> <CHILD_PATTERN_ID_0> <CHILD_PATTERN_ID_1>` ...

Both `<PARENT_PATTERN_ID>` and `<CHILD_PATTERN_ID_X>` are in the *LEVEL-NODE* form described in Section 2.2.1, and represent the patterns that were discovered by the algorithm.

Figure 3(a) shows an example of PC-list file. In this example, the patterns were generated using a constant minimum support and for this reason the PC-list can be used to create the frequent itemset lattice shown in Figure 3(b).

2.3 Examples

Figure 4 shows an example of using LPMiner. In this example, LPMiner is given an input transaction file `TranFile` shown in Figure 5(a). This transaction file has a total of 10 transactions. The "-s" option is used to set the constant minimum support to 50%. Thus, the output frequent pattern file `TranFile.fp` shown in Figure 5(b) contains frequent patterns that are supported by at least $10 \times 50/100 = 5$ transactions. For example, frequent pattern {`1 2 4`} has 6 supporting transactions–a support value of 60.0%. The first field in each one of the lines in the frequent pattern file, *e.g.*, `2-0`, indicates the PATTERN_ID of the pattern shown on that line.

In Figure 4, the "-p" option was used to generate a PC-list file. Figure 5(c) shows the PC-list file `TranFile.pc`. Each line of the PC-list file has the PATTERN_ID of the parent pattern followed by a set of PATTERN_IDs corresponding to its children patterns. For example, pattern {`2 4`} with PATTERN_ID `1-3` has children patterns {`2`} and {`4`} that have PATTERN_IDs {`0-6`} and {`0-4`}, respectively. This parent-children relationship is shown in the fifth line of Figure 5(c). Pattern {`2 4`} is also a child of pattern {`1 2 4`} with PATTERN_ID {`2-0`}. The first line of Figure 5(c) shows this relationship.

If LPMiner is given maximal pattern option "-x", all the non-maximal frequent patterns are eliminated from frequent patterns in Figure 5(b). Figure 5(d) shows the resulting frequent pattern file.

```
prompt% lpminer -s 50.0 -p TranFile
********************************************************************************
lpminer (PAFI 1.0) Copyright 2003, Regents of the University of Minnesota

Transaction File Information ---------------------------------------------------
  Transaction File Name:                        TranFile
  Number of Input Transactions:                 10
  Number of Distinct Items:                     21
  Average Number of Items In a Tran:            8.800
  Maximum Number of Items In a Tran:            12

Options ------------------------------------------------------------------------
  Minimum Output Pattern Size:                  0
  Maximum Output Pattern Size:                  4294967295
  Constant Minimum Support:                     50.000000
  PC List File Generation:                      Generate
  Non-Maximal Frequent Pattern Pruning:         Skip

Solution -----------------------------------------------------------------------
  Frequent Pattern File:                        TranFile.fp
  PC List File:                                 TranFile.pc
  Number of Frequent Patterns:                  14
  Number of Frequent Patterns[Length    1]      7
  Number of Frequent Patterns[Length    2]      6
  Number of Frequent Patterns[Length    3]      1

Timing Information -------------------------------------------------------------
  Input File Transformation:                    0.006 sec
  Generating Frequent Pattern File:             0.006 sec
  Generating PC List File:                      0.005 sec
********************************************************************************
```

Figure 4: Output of LPMiner

```
1 2 4 6 7 9 11
1 2 3 4 6 7 8 9 12 13 15 17
5 7 9 10 11 13 15 18
1 2 4 5 16 19 20
0 1 2 4 8 9 11 13 15
0 1 2 6 12 14 19
2 3 6 8 9 10 12 14 15 16 19
2 4 6 7 8 11 12 14 16 17
1 2 3 4 5 8 10 14 18
0 1 2 4 6 9 11 13
```

(a) Transaction File "TranFile"

```
0−0 5 50.000 11
0−1 5 50.000 8
0−2 6 60.000 9
0−3 6 60.000 6
0−4 7 70.000 4
0−5 7 70.000 1
0−6 9 90.000 2
1−0 5 50.000 2 8
1−1 5 50.000 2 9
1−2 6 60.000 2 6
1−3 7 70.000 2 4
1−4 6 60.000 1 4
1−5 7 70.000 1 2
2−0 6 60.000 1 2 4
```

(b) Frequent Pattern File "TranFile.fp"

```
2−0 1−3 1−4 1−5
1−0 0−1 0−6
1−1 0−2 0−6
1−2 0−3 0−6
1−3 0−4 0−6
1−4 0−4 0−5
1−5 0−6 0−5
0−0
0−1
0−2
0−3
0−4
0−5
0−6
```

(c) PC−List File "TranFile.pc"

```
0−0 5 50.000 11
1−0 5 50.000 2 8
1−1 5 50.000 2 9
1−2 6 60.000 2 6
2−0 6 60.000 1 2 4
```

(d) Maximal Pattern File "TranFile.fp"

Figure 5: Examples of the Input/Output files used/produced by LPMiner.

3 The SLPMiner Program

Usage

SLPMiner [optional parameters] *SequentialTranFile*

Description

Used to find all frequent sequential patterns satisfying a constant or length-decreasing support constraint from a sequential transaction file.

Required Parameters

SequentialTranFile
The name of the file that stores the input sequential transactions from which frequent sequential patterns are to be found. The format of the sequential transaction file is described in Section 3.1.1.

Optional Parameters

`-s FLOAT, --support=FLOAT`
This parameter sets the constant minimum support in percentage. This value must be in the range of [0.0, 100.0]. The default value is 5.0%. Note that if the minimum support is set to 0.0, all the frequent patterns with at least one supporting transaction are output. If *-S string* option is specified, -s float option is ignored.

`-S FILE, --supptable=FILE`
This parameter specifies the file that stores the length-decreasing support constraint. The format of this file is described in Section 3.1.2. This parameter hides *-s float* option.

`-m INT, --minsize=INT`
This parameter sets the minimum length of frequent patterns to be output. The default value is one. If this parameter is set to a value greater than one, the amount of time required to find the frequent patterns will be decreased.

`-M INT, --maxsize=INT`
This parameter sets the maximum length of frequent patterns to be output. The default value is 4,294,967,295 (0xffffffff). If this parameter is set to a smaller value, then the amount of time taken for finding frequent patterns may be decreased.

`-x, --maximal`
Generates maximal frequent patterns only. The current version of SLPMiner does not contain any optimizations to actually reduce the amount of time required to find maximal patterns, and this parameter is used solely to limit the amount of information that is being output.

`-t, --tid-list`
Generates a file that stores for each pattern, the IDs of the input sequential transactions that supports it. These lists are referred to as TID-lists.

The TID-lists are stored in a file, which has the same name as the input file with file extension ".tid" added. The format of the TID-list file is described in Section 3.2.3.

`-p, --parent-children-list`
Generates a file that shows the parent-children relationships among frequent patterns. These relationships are referred to as *PC-list*. For each frequent pattern, p, its PC-list contains all frequent patterns c, such that c is a maximal frequent sub-pattern of p. Note that when SLPMiner is used with a constant support constraint, the PC-lists correspond to the frequent pattern lattice.

The PC-lists are stored in a file, which has the same name as the input file with file extension ".pc" added. The format of the PC-list file is described in Section 3.2.2.

`-b INT, --bufsize=INT`
This parameter sets the size of the internal I/O buffer in megabytes. The default value is 50. The performance of SLPMiner can be improved by increasing the size of the buffer. The buffer size must be at least as large as the longest sequence in the internal binary format. This value can be calculated

in terms of the maximum number of itemsets in a sequence, m, and the maximum number of items in a sequence, n, using the following formula:

$$\frac{20 + 4n + 4m}{1000 \times 1000} \text{ MB}$$

This value is also obtained from the log file of SLPMiner.

`-d PATH, --dir=PATH`

This parameter sets the directory to be used for storing intermediate *.lf.N files (described in Section 5). The default value is the directory where the program is invoked. These files are generated during the execution of SLPMiner and are deleted upon successful execution. The performance of SLPMiner can be improved by providing a path on a disk that is attached locally on the machine that SLPMiner runs (*i.e.*, you should not be using a network mounted disk).

`-h, --help`

Displays a short summary of the various options to the standard output.

OUTPUT

The discovered frequent patterns are stored in a file that has the same name as the input file with file extension ".fp" added. The format of this frequent pattern file is described in Section 3.2.1.

The SLPMiner program also generates a log-file called `slpminer.log`, which stores various statistics regarding the input file, the discovered frequent patterns, and the amount of time taken during the various stages of the computation. The same information is also printed on the standard output.

NOTE

The SLPMiner program uses a disk-based implementation and can scale to very large datasets. The primary constraint on the size of the datasets that it can process is the amount of free disk-space that is available for storing intermediate data files.

3.1 Input File Formats

SLPMiner takes as input two different files. The first is the *SequentialTranFile* that contains the sequential transactions from which the frequent patterns will be discovered, and the second file is the length-decreasing support constraint file (*SuppFile*) that indicates the different minimum support values associated with each pattern length. Note that the *SuppFile* is optional and is used only when the `-S` is specified. The format of these files is described in the rest of this section.

3.1.1 Sequential Transaction File

A sequential transaction file is an ASCII file in which each line represents a sequential transaction. A sequential transaction is represented as an ordered list of itemsets separated by `-1`. Each item can be any string except -1, which is the itemset separator. There are no restrictions on how the items are ordered within each itemset. Figure 6 shows some examples of different sequential transaction files.

If a line contains nothing except spaces or tabs, the line is ignored and it is not counted when determining the total number of transactions. Also, if an itemset contains duplicate items, then the duplicate occurrences of an item within that itemset are ignored. Ordering of the items within each itemset does not affect the arrangement of items in frequent pattern files generated by SLPMiner.

3.1.2 Length-Decreasing Support Constraint File

The format of the length-decreasing support constraint file is identical to that used by LPMiner and was described in Section 2.1.2. The only difference is that since a sequential pattern consists of a sequence of itemsets, its *length* is equal to the total number of items in all of its itemsets.

```
1 2 −1 7 9 11 −1 7 8 9 −1 1 9 13 17
1 3 −1 1 2 6 7 8 9 −1 12 13 17
9 −1 10 13 15 18 −1 5 9 −1 11 15 18
1 2 4 5 −1 7 19 −1 1 2 4 −1 6 7 9 11
0 1 4 −1 9 11 13 −1 15
0 1 2 6 7 14 19 −1 2 −1 8 9 12 −1 16 19
1 2 3 −1 7 8 −1 6 9 10 12 −1 16 19
2 4 6 7 8 −1 11 12 14 16 17 −1 0 1 6 9 11 13
1 2 4 −1 0 7 8 −1 11 14 2 −1 8 −1 14 18
0 1 2 4 6 9 10 13
```

(a)

```
pear −1 peach pear
apple orange −1 apple pear −1 pear
apple orange peach −1 orange
melon orange peach −1 orange
```

(b)

Figure 6: Examples of sequential transaction files accepted by SLPMiner. Note that the items can be arbitrary strings.

3.2 Output File Formats

Upon successful execution, SLPMiner outputs the discovered patterns into a file called the *frequent pattern file*. If the `-x` option has been specified, then this file contains only the maximal patterns, otherwise it contains all patterns that satisfy the minimum support constraint(s). In addition, if the user has specified the `-p` option, SLPMiner will generate the *PC-list file* that contains the parent-children relationships among frequent patterns. Similarly, if the user has specified the `-t` option, SLPMiner will generate the *TID-list file* that shows the sequential transactions that are supported by each pattern. The format of these files is described in the rest of this section.

3.2.1 Frequent Pattern File

The format of the frequent pattern file is identical to that used by LPMiner and was described in Section 2.2.1.

3.2.2 PC-List File

The format of the PC-list file is identical to that used by LPMiner and was described in Section 2.2.2.

3.2.3 TID List File

For each frequent pattern, the *TID-list file* (*.tid) shows the set of sequential transactions that support it (*i.e.*, the set of sequential transactions that contain the pattern). Note that “TID” stands for *transaction identifier*. Each line of a TID-list file shows the list of supporting transactions for a discovered frequent pattern. The format of each line is the following:

`<PATTERN_ID> <TID_0> <TID_1>` . . .

`<PATTERN_ID>` is the ID of a frequent pattern and the remaining `<TID_x>` entries are the list of the supporting transactions separated by white spaces. The format of `<PATTERN_ID>` is in the *LEVEL-NODE* form described in Section 2.2.1. The supporting transaction IDs `<TID_x>` are non-negative integers and correspond to the order that these transactions appeared in the input file. The first transaction's ID in the input file is 0, the second transaction's ID is 1 and so on. TIDs in each TID-list are sorted in ascending order of TID value. Figure 8(d) shows an example of a TID-list file.

3.3 Examples

Figure 7 shows an example of using SLPMiner. In this example, SLPMiner is given an input sequential transaction file `SeqTranFile` shown in Figure 8(a). This file has a total of 10 sequential transactions. In each sequence, itemsets are separated by “-1”. The “-S” option is used to specify the length-decreasing support constraint specified in file `func` shown in Figure 8(b). Each line in `func` specifies the minimum support for patterns with a length. For example, the first line in `func` sets the minimum support of frequent patterns with length 1 to 80%. As for the patterns with length more than 4, the minimum support of length 4 is applied. The output frequent pattern file `SeqTranFile.fp` shown in Figure 8(c) contains frequent patterns that are supported by at least $10 \times 50/100 = 5$ sequential transactions.

```
prompt% slpminer -S func -t SeqTranFile
********************************************************************************
slpminer (PAFI 1.0) Copyright 2003, Regents of the University of Minnesota

Sequential Transaction File Information ---------------------------------------
  Sequential Transaction File Name:             SeqTranFile
  Number of Input Transactions:                 10
  Number of Distinct Items:                     20
  Average Number of Items In a Tran:            11.200
  Average Number of Itemsets In a Tran:         3.500
  Average Number of Items In an Itemset:        3.200
  Maximum Number of Items In a Tran:            15
  Maximum Number of Itemsets In a Tran:         5
  Maximum Binary Transaction Size:              92 Bytes

Options -----------------------------------------------------------------------
  Buffer Size:                                  50.000000 MB
  Temporary File Directory:                     Current Directory
  Minimum Output Pattern Size:                  1
  Maximum Output Pattern Size:                  4294967295
  Length-Decreasing Support Constraint File:    func
  PC List File Generation:                      Skip
  TID List File Generation:                     Generate
  Non-Maximal Frequent Pattern Pruning:         Skip

Solution ----------------------------------------------------------------------
  Frequent Pattern File:                        SeqTranFile.fp
  TID List File:                                SeqTranFile.tid
  Number of Frequent Patterns:                  11
  Size of Projected Databases:                  0.007388 MB
  Number of Frequent Patterns[Length    1]      3
  Number of Frequent Patterns[Length    2]      2
  Number of Frequent Patterns[Length    3]      4
  Number of Frequent Patterns[Length    4]      2

Timing Information ------------------------------------------------------------
  Input File Transformation:                    0.006 sec
  Generating Frequent Pattern File:             0.107 sec
********************************************************************************
```

Figure 7: Output of SLPMiner

The TID-list option "-t" is used to generate the TID-list file `SeqTranFile.tid` that is shown in Figure 8(d). Each line corresponds to a frequent pattern and starts from the PATTERN_ID of the frequent pattern, followed by its TID list. For example, pattern {`1 -1 7 8`} has a PATTERN_ID `2-0` and is supported by the first, second, seventh, and eighth transactions in `SeqTranFile`. Thus, the TID-list of pattern `2-0` is {`0 1 6 8`} (TID numbers start from 0).

```
1 2 −1 7 9 11 −1 7 8 9 −1 1 9 13 17
1 3 −1 1 2 6 7 8 9 −1 12 13 17
9 −1 10 13 15 18 −1 5 9 −1 11 15 18
1 2 4 5 −1 7 19 −1 1 2 4 −1 6 7 9 11
0 1 4 −1 9 11 13 −1 15
0 1 2 6 7 14 19 −1 2 −1 8 9 12 −1 16 19
1 2 3 −1 7 8 −1 6 9 10 12 −1 16 19
2 4 6 7 8 −1 11 12 14 16 17 −1 0 1 6 9 11 13
1 2 4 −1 0 7 8 −1 11 14 2 −1 8 −1 14 18
0 1 2 4 6 9 10 13
```

(a) Sequential Transaction File "SeqTranFile"

```
1 80.0
2 60.0
3 40.0
4 30.0
```

(b) Length Decreasing Support File

```
0−0 9 90.000 1
0−1 8 80.000 2
0−2 9 90.000 9
1−0 6 60.000 1 −1 9
1−1 7 70.000 1 2
2−0 4 40.000 1 −1 7 8
2−1 4 40.000 1 2 −1 7
2−2 4 40.000 1 2 −1 9
2−3 4 40.000 1 2 −1 8
3−0 3 30.000 1 2 −1 7 −1 9
3−1 3 30.000 1 2 −1 7 8
```

(c) Frequent Pattern File "SeqTranFile.fp"

```
0−0 0 1 3 4 5 6 7 8 9
0−1 0 1 3 5 6 7 8 9
0−2 0 1 2 3 4 5 6 7 9
1−0 0 1 3 4 5 6
1−1 0 1 3 5 6 8 9
2−0 0 1 6 8
2−1 0 3 6 8
2−2 0 5 6 8
2−3 0 3 5 6
3−0 0 3 6
3−1 0 6 8
```

(d) TID−List File "SeqTranFile.tid"

Figure 8: Examples of the Input/Output files used/produced by SLPMiner.

4 The FSG Program

USAGE

fsg [optional parameters] *GraphTranFile*

DESCRIPTION

Finds all frequent connected undirected subgraphs satisfying a given minimum support threshold constraint.

REQUIRED PARAMETER

GraphTranFile
The name of the file that stores the input graph transactions from which frequent connected subgraphs are to be found. The format of the graph transaction file is described in Section 4.1.1.

OPTIONAL PARAMETERS

`-s FLOAT, --support=FLOAT`
This parameter sets the minimum support in percentile. This value must be in the range of (0.0, 100.0]. The default value is 5.0%. Note that if the minimum support is set to 0.0, all the frequent subgraphs with at least one supporting transaction are generated.

`-m INT, --minsize=INT`
This parameter sets the minimum size of frequent connected subgraphs to be generated. The size of a frequent subgraph is the total number of edges in it. For example, the sizes of a triangle and a square are three and four respectively. The default value is one.

`-M INT, --maxsize=INT`
This parameter sets the maximum size of frequent connected subgraphs to be generated. The default value is equal to INT_MAX defined in `<limit.h>` and 2,147,483,647 on typical 32-bit systems. If this parameter is set to a small value, the program will stop after finding all the frequent subgraphs of the specified size.

`-x, --maximal`
Generates only maximal frequent subgraphs. Because the current version of FSG does not contain any optimizations for getting the maximal patterns only, note that this option does not reduce the running time.

`-t, --tid-list`
Generates a file with the suffix ".tid" which contains the lists of supporting transaction IDs for each frequent subgraph discovered. These lists are referred to as *TID-lists*. The format of the TID-list file is described in Section 4.2.3.

`-p, --parent-children-list`
Generates a file that shows the parent-children relationships among frequent patterns. These relationships are referred to as *PC-list*. For each frequent pattern, *p*, its PC-list contains all frequent patterns *c*, such that *c* is a maximal frequent sub-pattern of *p*.

The PC-lists are stored in a file, which has the same name as the input file with file extension ".pc" added. The format of the PC-list file is described in Section 4.2.2.

`-h, --help`
Displays a short summary of the options to the standard output.

OUTPUT

The discovered frequent connected subgraphs are stored in a file with the same basename as the input file and with the suffix ".fp". The format of this frequent pattern file is described in Section 4.2.1.

NOTE

FSG is implemented as in-memory program. The size of the datasets that can be processed are limited by the amount of physical memory in your system.

4.1 Input File Format

FSG takes as input the *GraphTranFile* that contains the graph transactions from which the frequent patterns will be discovered. Basically this format is a simplified version of SUBDUE [2] graph file format. The detail of the format is described in the rest of this section.

4.1.1 Graph Transaction File

A graph transaction file is an ASCII file that contain a set of graph transactions. Each graph transaction begins with the transaction line which starts with the letter "`t`". Next, a set of vertex lines follows. Then, a set of edge lines comes. There are only four types of lines including comments which are shown in Table 1. Vertex IDs are non-negative integers assigned in the increasing order sequentially from zero. If a graph has N vertices, there should be N vertex lines whose "`<ID>`" is from 0 to $N-1$. If a graph has M edges, there should be M lines in total each of which contains a pair of vertex IDs as its endpoints.

Line type	Format	Explanation
Comment	`# ...`	Everything ignored after "#"
Transaction	`t`	The beginning of a transaction
Vertex	`v <ID> <LABEL>`	A vertex ID `<ID>` and its vertex label `<LABEL>`
		`<ID>` a non-negative integer, `<LABEL>` a string without spaces
Edge	`u <ID1> <ID2> <LABEL>`	Endpoints `<ID1>` and `<ID2>`, and its edge label `<LABEL>`
		`<ID1>`, `<ID2>` vertex IDs, `<ID1>` $<$ `<ID2>`
		`<LABEL>` a string without spaces

Table 1: Format of the graph transaction file

Figure 9 shows a graph of four vertices and five edges. Edges are unlabeled (*i.e.*, all edges have the same label "`E`") and vertex labels are either "`black`" or "`white`". Figure 10 shows the corresponding graph transaction. Note that vertex and edge lines must be sorted in the ascending order based on the vertex IDs and each edge appears only once because "`<ID1>`" must be smaller than "`<ID2>`" in the edge line.

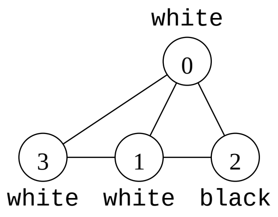


Figure 9: Sample graph

```
t # 4 vertices and 5 edges
v 0 white
v 1 white
v 2 black
v 3 white
u 0 1 E
u 0 2 E
u 0 3 E
u 1 2 E
u 1 3 E
```

Figure 10: Sample graph transaction

4.2 Output File Formats

Upon successful execution, FSG outputs the discovered patterns into a file called the *frequent pattern file*. If the `-x` option has been specified, then this file contains only the maximal patterns, otherwise it contains all patterns that satisfy the minimum support constraint. In addition, if the user has specified the `-p` option, FSG will generate the *PC-list file* that contains the parent-children relationships among frequent patterns. Similarly, if the user has specified the `-t` option, FSG will generate the *TID-list file* that shows the graph transactions that are supported by each pattern. The format of these files is described in the rest of this section.

4.2.1 Frequent Pattern File

The format of the frequent-pattern file (*.fp) that FSG uses to output the patterns that it discovered is similar to that used for specifying the input graph transactions (described in Section 4.1.1). This is because FSG's output is a set of frequent subgraphs which can also be regarded as a set of graphs.

The only format difference is that the frequent-pattern file contains additional information about the different patterns such as their pattern ID and their frequencies. This information appear as comments in each "Transaction" line (*i.e.*, the line starts with "`t`"). The exact format of the "Transaction" line of the frequent-pattern file and its meaning is shown in Table 2.

Line type	Format	Explanation
Transaction	`t # <PATTERN_ID>, <COUNT>`	The beginning of a transaction, i.e., a frequent subgraph. `<PATTERN_ID>` is the identifier assigned to the subgraph. This identifier is in a similar *LEVEL-NODE* format as that used by LPMiner and SLPMiner. However, in the case of FSG, *LEVEL* corresponds to the number of edges in the frequent subgraph. The same ID is used by the PC-List and TID-List files. `<COUNT>` shows the total number of transactions which contain at least one instance of the frequent subgraph. Naturally this count should be greater or equal to the support threshold specified by a user.

Table 2: Format of the "Transaction" line of the frequent-pattern file.

4.2.2 PC-List File

The format of the PC-list file is identical to that used by LPMiner and was described in Section 2.2.2.

Figure 11 shows a frequent subgraph of size 5 as well as its 5 subgraphs of size 4. Note that there are five ways to obtain size-4 subgraphs from this size-5 subgraph by removing each edge, only three are distinct in terms of isomorphism. The corresponding line in the PC-list file becomes "5-0 4-0 4-1 4-2" showing that the subgraph 5-0 contains the three distinct subgraphs 4-0, 4-1 and 4-2.

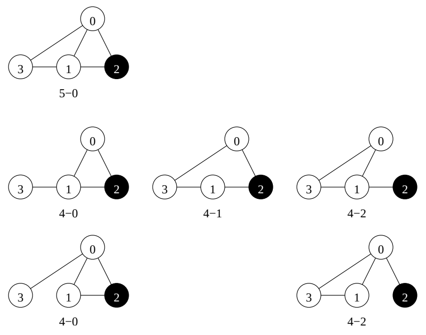


Figure 11: Sample subgraph of size 5 and its children of size 4

```
5-0 4-0 4-1 4-2
```

Figure 12: The corresponding line in the PC-list file

4.2.3 TID-List File

The format of the TID-list file is identical to that used by SLPMiner and was described in Section 3.2.3.

4.3 Examples

Figure 13 shows the execution of FSG at the command line and its output to the standard output. The input file is named "test.g" which is shown in Figure 14. The discovered frequent patterns are stored in the file named "test.fp"

```
user@machine:~ 1 $ ./fsg -s 100.0 -pt test.g
*******************************************************
fsg 1.34 (PAFI 1.0) Copyright 2003, Regents of the University of Minnesota

Transaction File Information --------------------------
  Transaction File Name:                      test.g
  Number of Input Transactions:               2
  Number of Distinct Edge Labels:             1
  Number of Distinct Vertex Labels:           2
  Average Number of Edges In a Transaction:   4
  Average Number of Vertices In a Transaction: 4
  Max Number of Edges In a Transaction:       6
  Max Number of Vertices In a Transaction:    4

Options -----------------------------------------------
  Min Output Pattern Size:                    1
  Max Output Pattern Size:                    2147483647(INT_MAX)
  Min Support Threshold:                      100.0% (2 transactions)
  Generate Only Maximal Patterns:             No
  Generate PC-List:                           Yes
  Generate TID-List:                          Yes

Outputs -----------------------------------------------
  Frequent Pattern File:                      test.fp
  PC-List File:                               test.pc
  TID-List File:                              test.tid

  Number of Size-1  Frequent Patterns:        1
  Number of Size-2  Frequent Patterns:        1
  Number of Size-3  Candidates:               3
  Number of Size-3  Frequent Patterns:        1

  Largest Frequent Pattern Size:              3
  Total Number of Candidates Generated:       3
  Total Number of Frequent Patterns Found:    3

Timing Information ------------------------------------
  Elapsed User CPU Time:                      0.0[sec]
*******************************************************
```

Figure 13: Execution of FSG

and is shown in Figure 17. Note that besides the actual discovered subgraphs, the frequent pattern file also contains a log of the run as comment fields. The options "-p" and "-t" are used each of which specifies to generate the files "test.pc" and "test.tid" respectively. The contents of the two files are shown in Figures 15 and 16.

```
t # triangle
v 0 V
v 1 V
v 2 V
u 0 1 E
u 0 2 E
u 1 2 E
t # K4
v 0 V
v 1 V
v 2 V
v 3 W
u 0 1 E
u 0 2 E
u 0 3 E
u 1 2 E
u 1 3 E
u 2 3 E
```

Figure 14: Graph Transaction File `test.g`

```
1-0
2-0 1-0
3-0 2-0
```

Figure 15: PC-List File `test.pc`

```
1-0 0 1
2-0 0 1
3-0 0 1
```

Figure 16: PC-List File `test.tid`

```
# ********************************************************
# fsg 1.34 (PAFI 1.0) Copyright 2003, Regents of the University of Minnesota
#
# Transaction File Information -------------------------
#   Transaction File Name:                      test.g
#   Number of Input Transactions:               2
#   Number of Distinct Edge Labels:             1
#   Number of Distinct Vertex Labels:           2
#   Average Number of Edges In a Transaction:   4
#   Average Number of Vertices In a Transaction: 4
#   Max Number of Edges In a Transaction:       6
#   Max Number of Vertices In a Transaction:    4
#
# Options ----------------------------------------------
#   Min Output Pattern Size:                    1
#   Max Output Pattern Size:                    2147483647(INT_MAX)
#   Min Support Threshold:                      100.0\% (2 transactions)
#   Generate Only Maximal Patterns:             Yes
#   Generate PC-List:                           Yes
#   Generate TID-List:                          Yes
#
# Outputs ----------------------------------------------
#   Frequent Pattern File:                      test.fp
#   PC-List File:                               test.pc
#   TID-List File:                              test.tid
#
t # 1-0, 2
v 0 V
v 1 V
u 0 1 E
t # 2-0, 2
v 0 V
v 1 V
v 2 V
u 0 1 E
u 0 2 E
t # 3-0, 2
v 0 V
v 1 V
v 2 V
u 0 1 E
u 0 2 E
u 1 2 E
#   Number of Size-1  Frequent Patterns:        1
#   Number of Size-2  Frequent Patterns:        1
#   Number of Size-3  Candidates:               3
#   Number of Size-3  Frequent Patterns:        1
#
#   Largest Frequent Pattern Size:              3
#   Total Number of Candidates Generated:       3
#   Total Number of Frequent Patterns Found:    3
#
# Timing Information -----------------------------------
#   Elapsed User CPU Time:                      0.0[sec]
# ********************************************************
```

Figure 17: FP File `test.fp`

5 Filename Rules

PAFI's stand-alone programs use a specific file extension for each file type as shown in Table 3. In the table, we assume the input transaction, sequential transaction, or graph transaction file is TEST, but you can choose any file name. By default, LPMiner, SLPMiner, and FSG generate those files in Table 3 on the directory where the input transaction file resides. Only one exception is *.lf.N files that can be put in a directory of your choice. This is because these files are accessed so extensively that the I/O performance on these files affects the amount of time taken by SLPMiner dramatically.

Note that if a file with the same name exists, LPMiner, SLPMiner, and FSG will overwrite it without asking you. Therefore, it is safe to avoid putting files that have the same prefix as the input transaction file. Upon a successful termination, all temporary files are deleted.

File Name	Input/Output	Description	Option	Program
`TEST`	Input	Item/Sequence/Graph Transaction file	N/A	LPMiner, SLPMiner, FSG
`TEST.fp`	Output	Frequent pattern file	N/A	LPMiner, SLPMiner, FSG
`TEST.pc`	Output	PC list file	`-p`	LPMiner, SLPMiner, FSG
`TEST.tid`	Output	TID list file	`-t`	SLPMiner, FSG
`TEST.t_conv`	Temporary	Converted transaction file	N/A	LPMiner, SLPMiner
`TEST.fp_conv`	Temporary	Converted frequent pattern file	N/A	LPMiner, SLPMiner
`TEST.slp`	Temporary	Binary transaction file	N/A	SLPMiner
`TEST.lf.`N	Temporary	Projected database file ($N = 0, 1, 2, \ldots$)	N/A	SLPMiner

Table 3: Extensions of files derived from transaction file "TEST".

6 System Requirements and Contact Information

PAFI is written in ANSI C and C++ and has been extensively tested under Linux and Solaris. At this point PAFI's distribution is only in a binary format, as it is actively under development. However, we expect to make the source code available in future releases.

Even though, PAFI contains no known bugs, it does not mean that all of its bugs have been found and fixed. If you find any problems, please send email to *karypis@cs.umn.edu*, with a brief description of the problem you have found. Also, any future updates to PAFI will be made available on WWW at *http://www.cs.umn.edu/˜pafi*.

7 Copyright Notice and Usage Terms

8 References

[1] Mukund Deshpande and George Karypis. Automated approaches for classifying structure. In *Proceedings of the 2nd ACM SIGKDD Workshop on Data Mining in Bioinformatics*, 2002.

[2] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[3] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *IEEE International Conference on Data Mining*, 2001. Also available as a UMN-CS technical report, TR# 01-028.

[4] Michihiro Kuramochi and George Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, (in press), 2003.

[5] Masakazu Seno and George Karypis. Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *IEEE International Conference on Data Mining*, 2001. Also available as a UMN-CS technical report, TR# 01-026.

[6] Masakazu Seno and George Karypis. Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *IEEE International Conference on Data Mining*, 2002.